(1986. 11. 11 受理)

データベースのロック割当て方式の解析

西　垣　　通

An Analysis of Database Lock Scheduling Strategies

Tohru NISHIGAKI

Synopsis. Strategies for lock scheduling in a database system are discussed. A lock must be set in some mode when a database resource is accessed by a transaction, in order to prevent inconsistencies. This may cause a deadlock or significant queueing delays. In this paper a general model of wait-for graph is first developed. The model makes it possible to detect deadlocks for a multi-mode lock system, including artificial deadlocks. Though these artificial deadlocks may frequently occur under ordinary lock scheduling strategies, they have never been detected by the conventional wait-for graph model. Secondly, three lock scheduling strategies: First In First Out (FIFO), EXPediency (EXP), and First In COmpatible First Out (FICOFO), are introduced and compared. Theoretical reasoning and simulations revealed the lowest and the highest deadlock frequency for EXP and FIFO, respectively. The deadlock frequency of FICOFO was between the two. In addition, EXP and FIFO each yielded the smallest and the largest queueing delays, for locks of high compatibility mode. However, for locks of low compatibility mode, significant queueing delays were observed in EXP, as compared with FICOFO and FIFO.

1. まえがき

データベース・システムにおいては，並行に処理される複数のトランザクションが同一のデータベース・リソースを読み書きする可能性がある。したがってシステムの論理的一貫性を保つため，通常，ロックによるリソース・アクセス制御が行われる。すなわち，トランザクションは或るリソースをアクセスするに先立ち，当該リソースのロックを要求し，これを獲得しなければならない。既に当該リソースのロックが他のトランザクションにより保持されておりこれを獲得できない場合，このトランザクションはロック待ち状態となる。多くのデータベース・システムでは，トランザクションはその処理進行にともなって次々とロックを獲得してデータをアクセスし，いったん獲得したロックはトランザクションの処理完了まで保持される[1)4)]。したがって，複数のトランザクションが互いに相手の保持するロックを待ち合い，これらの処理が進行不能となる「デッドロック現象」が発生する恐れがある。デッドロックが発生した場合ただちにこれを検知し，ロック待ち状態にあるトランザクション群のうちいずれかを強制終

了させてその保持するロックを全て解放し，デッドロックを解消することが必要である。

従来，デッドロック検知は，システム内のトランザクションとリソースをノードとし，これらをロック待ちおよびロック保持関係により結合した待ちグラフ（wait-for graph）を用いて行われてきた[1)4)5)6)7)]。例えば，リソース r_1，r_2 のロックがそれぞれトランザクション t_1，t_2 により保持されているとき，この関係を $r_1 \rightarrow t_1$，$r_2 \rightarrow t_2$ という割当てエッジ（assignment edge）で表現する。ここで t_1，t_2 がそれぞれ r_2，r_1 のロックを要求したとすると，これは $t_1 \rightarrow r_2$，$t_2 \rightarrow r_1$ という要求エッジ（request edge）で表わされる。明らかにこのとき，この待ちグラフは図 1(a) のようにサイクルを形成し，デッドロックが発生したとみなされる。

しかし，上記のようなデッドロック検知法は，実システムでは必ずしも適切とは言えない。この理由として，まず，トランザクションの要求するロックのモードが考慮されていないことがあげられる。実システムでは，通常，リソースを読みこむときは S(Share) モード，リソースに書き出すときは X (Exclusive) モードでロックを要求する。X モードのロックを保持できるトランザクションは 1 個のみであるが，S モードのロックは同時に複数のトランザクションにより保持されうる。いいかえると，X モードと X モード，X モードと S モードのロックは「非両立」であるが，S モードと S モードのロックは「両立」する。したがって，図 1(a) の例でも t_1 が r_1 の S モードのロックを保持しているとき，t_2 が r_1 の S モードのロックを要求したとすれば，この要求は受けつけられるのでデッドロックは発生しない。

複数モードの場合のデッドロック検知の問題は，Rypka と Lucido により扱われた[2)]。従来の待ちグラフは拡張され，ロックの保持（割当て）ないしロックの要求を表わすエッジにそのモードを表わすラベルが付加される。このようなエッジがサイクルを形成するとき，サイクル中の全ての要求エッジについて，そのモードとこれに隣接する割当てエッジのモードが非両立ならば，このサイクルを D サイクルとよぶ。Rypka と Lucido は，複数モードのロックを用いるシステムのデッドロック検知は D サイクルの検知により遂行できると論じた。

しかし，実システムでは，D サイクルによっては検知できない「人為デッドロック（artificial deadlock)」が発生する場合がある[3)]。人為デッドロックとは，ロック割当て方式に依存

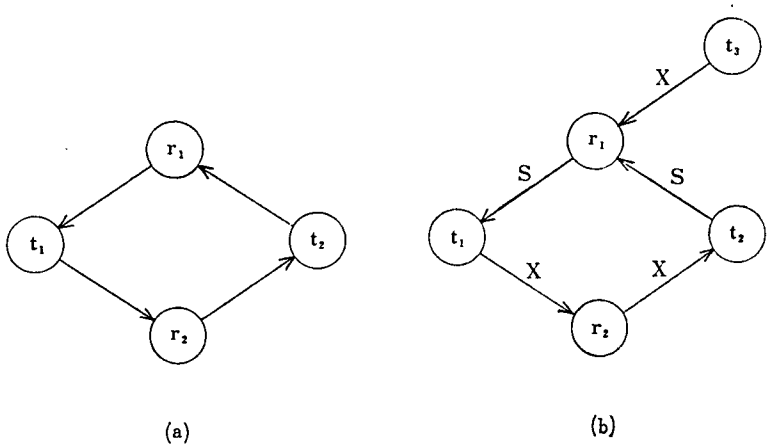


図 1　デッドロック検知のための待ちグラフ

して発生するものである。例えば，実システムでは，或るリソースに関するロックの割当て要求を FIFO (First In First Out) 方式すなわち先着順に受けつけることが多い。このときいま，リソース r_1 の S モード・ロックをトランザクション t_1，リソース r_2 の X モード・ロックをトランザクション t_2 がそれぞれ保持し，また，r_1 の X モード・ロックをトランザクション t_3，r_2 の X モード・ロックを t_1 が，それぞれ要求して待ち状態にあるとする。ここで t_2 が新たに r_1 の S モード・ロックを要求したとする。このとき図 1(b) に示すサイクルは，$t_2 \rightarrow r_1$ の要求エッジのモードと $r_1 \rightarrow t_1$ の割当てエッジのモードが両立するので D サイクルではない。しかし，ロック割当て方式が FIFO の場合，t_3 の r_1 に対する X モードのロック要求が受けつけられない限り t_2 の r_1 に対する S モードのロック要求も受けつけられないので，実際にはデッドロックが発生する。これは，ロック割当て方式が FIFO であることから生じた人為デッドロックである。

人為デッドロックも実システムにおいては問題となるので，これを検知する必要がある。このためには，ひとつのリソースに関して，そのロックを保持中のトランザクションとロックを要求中のトランザクションとの関係のみならず，ロックを要求中のトランザクションどうしの関係（例えば本例では t_2 と t_3 との関係）をも待ちグラフに反映させる必要がある。

以上の考察に鑑み，本論文の目的は次の 2 点とする。

(1) デッドロック検知のための一般的方法を与えること。具体的には，待ちグラフの概念を拡張し，そこでのサイクル形成が人為デッドロックをも含んだ一般的なデッドロックの発生に対応するように，より広い定義を与えること。

(2) 定義したグラフを用いて，ロック割当て方式の比較解析を行う。比較する方式としては，実システムで多用される FIFO, EXP (EXPediency) に加え FICOFO (First In COmpatible First Out) を採択する。EXP とは，ロック要求が到着したとき，そのモードが現在既に割り当てられている当該リソースのロックのモードと両立するかぎり，到着順序によらずこれを受けつけるという方式である。EXP のもとでは人為デッドロックは発生しない。また FICOFO とは，本論文で新たに導入する方式であり，ロック要求が到着したとき，既に割り当てられているか要求待ち状態にある全ての当該リソースのロックのモードと両立する場合に限って，このロック要求を受けつけるという方式である。

2. デッドロックとその検知

2.1 用語の定義

ロック割当て方式とデッドロックの問題を論ずるための基本的用語を以下に定義する。

データベース・システムにおける論理的に独立した処理単位を「トランザクション」と呼ぶ。トランザクションはその実行にあたり，データベース内の「リソース」を使用する。データベースのページはリソースの例である。トランザクションはリソースの使用に際して「ロック」を獲得しなければならない。ロックの種別を「モード」とよぶ。モードによってトランザクションによるリソースの使用法が定められる。S (Share), X (Exclusive) などはモードの例で

ある。いま，二つのトランザクションが或るリソースのロックを要求したとき，これらが共に受けつけられるためには，両者のロックのモードが「両立」しなければならない。2変数からなる両立性関数は以下のように定義される。

$$f(m_1, m_2)=\begin{cases}1：モード\ m_1\ と\ m_2\ が両立\\0：上記以外\end{cases} \quad \cdots\cdots(1)$$

例えば，通常

$$\begin{cases}f(S,\ S)=1\\f(X,\ S)=f(X,\ X)=0\end{cases} \quad \cdots\cdots(2)$$

である。モード体系（モードの数，両立性関数値など）は，システムの論理的一貫性を保ちつつ性能を向上させるように，システムにおいて定められる[1]。

トランザクションが或るリソースを使用するために或るモードでロックを要求したとき，そのモードと非両立なモードでこのリソースのロックを既に保持している他のトランザクションが存在すると，当該トランザクションは「待ち (wait/block) 状態」となる。ただし，このような他のトランザクションが存在しない場合でも，当該トランザクションのロック要求は必ずしも受けつけられるとは限らない。モード両立という必要条件をみたすトランザクションのロック要求の中で，どれを受けつけるかの選択基準を与えるのが「ロック割当て方式 (lock scheduling strategy)」である。FIFO はその一例である（FIFO の詳細は3.1節参照)。

「デッドロック」とは，いくつかの待ち状態のトランザクションが存在し，現在待ち状態に無いトランザクションに対するリソースの割当て操作/割当て解除操作をいかに行っても，これらのトランザクションの待ち状態を終了させることが不可能な状態のことである。デッドロックの中で，その発生の有無がロック割当ての方式に左右される様なものを「人為デッドロック」とよぶ。また，デッドロック以外の原因で，トランザクションのリソース使用のためのロック要求が長時間にわたって受けつけられない現象を「沈みこみ」とよぶ。

2.2 待ちグラフとデッドロック

人為デッドロックを含み，一般にデッドロックを検知するための待ちグラフ(wait-for graph)を以下に定義する。さらに Rypka と Lucido の定義した D サイクル[2]，また従来の待ちグラフ[1]と，本定義との関連についても言及する。

或るリソース r を使用するため，トランザクション t がモード $m(t)$ のロックを要求したとする。このとき r について，既にロックを保持しているトランザクションの集合を H とする。

$$H=\{h_1,\ h_2,\ h_3,\ \cdots\} \quad \cdots\cdots(3)$$

トランザクション $h(\epsilon H)$ の保持するロックのモードを $m(h)$ と書く。また，t より前に r のロックを要求したが受けつけられず待ち状態にあるトランザクションの集合を W とする。

$$W=\{w_1,\ w_2,\ w_3,\ \cdots\} \quad \cdots\cdots(4)$$

トランザクション $w(\epsilon W)$ の要求するロックのモードを $m(w)$ と書く。いま，或るトランザクション $h(\epsilon H)$ について，

$$f(m(h),\ m(t))=0 \quad \cdots\cdots(5)$$

ならば，「t は h を待つ」と定義し，待ちグラフ中で「$t\rightarrow h$」なる有向エッジを書く。さらに，所与のロック割当て方式のもとで，或るトランザクション $w(\epsilon W)$ のロック要求が受けつけられない限り t のロック要求も受けつけられないとき，「t は w を待つ」と定義し，待ちグラフ中で「$t\rightarrow w$」なる有向エッジを書く。ここで，この関係の成立はロック割当て方式に依存しており，$m(t)$ と $m(w)$ は必ずしも非両立とは限らない。両立する場合は，w のロック要求が受けつけられさえすればただちに t のロック要求も受けつけられる可能性がある。以下に例を示す。

（例）

$H=\{h_1\}$, $W=\{w_1\}$, $m(h_1)=X$, $m(t)=S$, とする。このとき，ロック割当て方式が FIFO なら「$t\rightarrow w_1$」なる有向エッジが待ちグラフ中に形成される。しかし式 (2) より，

$$f(m(w_1),\ m(t))=f(S,\ S)=1$$

である。h_1 がロックを解放すれば，w_1 と t は同時にロックを保持できる。

（例終り）

システム内の各トランザクションをノードとし，各リソースについて上記のような待ち関係を有向グラフで表わすことにより，待ちグラフ (wait-for graph) が書ける。ここで，待ちグラフにおいてサイクルが形成されることがデッドロック発生の必要十分条件である。

上記のように定義した待ちグラフにおいて，$t\rightarrow w$ なる有向エッジを取り除き，$t\rightarrow h$ なる有向エッジのみから構成されるパスが，Rypka と Lucido の定義した D パスに他ならない。また，このパスがサイクルをなすとき，D サイクルとなる[2]。D サイクルは，ロック割当て方式によらず発生するデッドロックに対応する。D サイクル以外のサイクル，すなわちサイクル中に $t\rightarrow w$ なる有向エッジを少なくとも 1 個含むサイクルは，ロック割当て方式によって発生する人為デッドロック (artificial deadlock) である[3]。

一般に待ちグラフ中のトランザクション t_1, t_2, t_3 について，$t_1\rightarrow t_2$ かつ $t_2\rightarrow t_3$ のとき，$t_1\rightarrow t_3$ なる関係が導かれる。以下これを「リダクション」と呼ぶ。デッドロックとは，リダクションの結果いずれかのトランザクションについて $t\rightarrow t$ なる関係が成り立つことに他ならない。

いま，t から有向エッジをたどって到達可能な w，すなわち，必要ならリダクションを実行して，$t\rightarrow w$ なる有向エッジが得られるような w の集合を W_R とする。ここで，いかなる $w(\epsilon W_R)$ についても $w\rightarrow t$ なる有向エッジが存在しないようなロック割当て方式を「自然な」ロック割当て方式と定義する。以下，本論文で扱うロック割当て方式は全て自然なロック割当て方式であると仮定する。（実用上意味のあるロック割当て方式は，3.1 節で提示する三方式を含め，全て自然なロック割当て方式である。自然でないロック割当て方式として，「トランザクションを三つのグループ G_1, G_2, G_3 に分け，G_1, G_2, G_3 グループのトランザクション

はそれぞれ G_3, G_1, G_2 グループのトランザクションより優先的にロックを割り当てる」などの方式を仮想的に設定することはできる。しかし，これらはデッドロックを多発させるだけで実用上無意味である。）ここで次の命題が導かれる。

〔命題 1〕

トランザクション t を含むサイクルは，リソース r のロック保持中のトランザクション集合 H の元を少なくとも 1 個含む。

（証　明）

$h(\epsilon H)$ を全く含まないサイクルが存在すると仮定する。トランザクションはロック要求を受けつけられないかぎり処理を続行できない。したがって，或る時点で，トランザクションがロック要求待ちとなるリソースはたかだか 1 個である。すなわち，t や $w(\epsilon W)$ を始点とする有向エッジは，r に関する $t\to h$, $t\to w$, $w\to h$, $w\to w'(w'\epsilon W,\ w\neq w')$ の 4 種類のみである。したがって，h を含まず，t から始まるサイクルがあるとすれば，それは t と W_R の元からなるサイクルのみである。しかし自然なロック割当て方式のもとでは，定義よりこのようなサイクルは存在しない。

（証明終り）

次に，リソース r に関してえられる有向エッジについて必要ならリダクションを行い，$t\to h$ なる有向エッジがえられるような h (ϵH) の集合を H_R とする。ここであらためて命題 1 を書き直し，命題 1′ を得る（証明は明らかなので略）。

〔命題 1′〕

トランザクション t がリソース r のロックを要求したとき，デッドロックが発生したとすると，そのサイクル中には H_R の元が含まれる。

また，式 (5) が成立する $h(\epsilon H)$ の集合，すなわちリダクションを行うことなく $t\to h$ なる有向エッジが存在するような h の集合を H_D とする。当然

$$H_R\supseteq H_D \qquad \cdots\cdots(6)$$

である。ここで次の命題が得られる。

〔命題 2〕

トランザクション t がリソース r のロックを要求したとき，デッドロックが発生したとする。このサイクル中に H_D の元でない $h(\epsilon H_R)$ が含まれるならば，このデッドロックは人為デッドロックである。

（証　明）

もし，このデッドロックが人為デッドロックでないとすると，対応するサイクルは D サイクルである。D サイクルの定義より，サイクル中に含まれる $h(\epsilon H_R)$ について，

$$f(m(h),\ m(t))=0$$

でなければならない。したがって h は H_R の元となり，仮定に反する。

（証明終り）

命題2より，差集合 $\{H_R-H_D\}$ について，

$$\{H_R-H_D\}\fallingdotseq\phi \quad \cdots\cdots(7)$$

のとき，人為デッドロックが発生する可能性がある。式(7)の例を示す。

(例)

$H=\{h_1\}$，$W=\{w_1\}$，$m(h_1)=S$，$m(t)=S$，ロック割当て方式は FIFO とすると，

$H_D=\phi$，$H_R=\{h_1\}$，$\{H_R-H_D\}=\{h_1\}$

である。

(例終り)

なおここで特に，全てのリソースとトランザクションについて，その要求ロック・モードが X であるとする。このとき，リソースのロックを保持しうるトランザクションは高々1個であり，常に

$$H_R=H_D \quad \cdots\cdots(8)$$

である。この場合，人為デッドロックは発生しない。これは従来の待ちグラフ[1)4)5)6)7)]で検出されるデッドロックに対応する。

3. ロック割当て方式

3.1 FIFO, EXP, FICOFO

実システムで多用されるロック割当て方式として FIFO (First In First Out)，EXP (EXPediency) がある。本節では，これに FIFO を一部変更した FICOFO (First In COmpatible First Out) を加え，三つの方式を提示するとともに，各方式のデッドロック発生率について考察する。

(1) FIFO

先着時にロック要求を受けつける方式である。

(a) ロック要求到着時処理

トランザクション t がリソース r にモード $m(t)$ のロックを要求したとき，式(9)もしくは式(10)が成立すればこの要求を受けつける。いずれも成立しないとき，ロック要求は待行列の最後尾に並ぶ。

$$H=W=\phi \quad \cdots\cdots(9)$$

$$\begin{cases} f(m(h),\ m(t))=1 \quad \text{for} \quad \forall h\in H \\ W=\phi \end{cases} \quad \cdots\cdots(10)$$

(b) ロック解放時処理

トランザクション t がリソース r のロックを解放したとき，待行列の先頭から順にロ

ック要求を受けつけられるか否か調べる。ロック要求は，既に受けつけた全要求のモードと両立するかぎり受けつけられる。非両立なモードのロック要求が現れると，本操作は終了し，それ以降に並んだロック要求は待ち状態にとどまる。

(2) EXP

到着順序によらず，両立するかぎりロック要求を受けつける方式である。

(a) ロック要求到着時処理

トランザクション t がリソース r にモード $m(t)$ のロックを要求したとき，式 (9) もしくは式(11)が成立すればこの要求を受けつける。いずれも成立しないとき，ロック要求は待行列の最後尾に並ぶ。

$$f(m(h),\ m(t))=1 \quad \text{for} \quad \forall h\in H \qquad \cdots\cdots(11)$$

(b) ロック解放時処理

トランザクション t がリソース r のロックを解放したとき，待行列の先頭から順にロック要求を受けつけられるか否か調べる。ロック要求は，既に受けつけた全要求のモードと両立するかぎり受けつけられる。本操作は，原則として待行列の最後尾まで実行される（実際にはその必要が無い場合もある。3.2節参照)。

(3) FICOFO

既に到着したロック要求と両立するかぎり，ロック要求を受けつける方式である。

(a) ロック要求到着時処理

トランザクション t がリソース r にモード $m(t)$ のロックを要求したとき，式 (9) もしくは式(12)が成立すればこの要求を受けつける。いずれも成立しないとき，ロック要求は待行列の最後尾に並ぶ。

$$\begin{cases} f(m(h),\ m(t))=1 \quad \text{for} \quad \forall h\in H \\ f(m(w),\ m(t))=1 \quad \text{for} \quad \forall w\in W \end{cases} \qquad \cdots\cdots(12)$$

(b) ロック解放時処理

トランザクション t がリソース r のロックを解放したとき，待行列の先頭から順にロック要求を受けつけられるか否か調べる。ロック要求は，既に受けつけた全要求のモードと両立し，かつ，当該要求より前に到着して（待行列中で前に位置して）受けつけられず待ち状態にある全要求のモードと両立するかぎり受けつけられる。本操作は，原則として待行列の最後尾まで実行される（実際にはその必要が無い場合もある。3.2節参照)。

FIFO, EXP, FICOFO の三方式を比べると，FIFO では，t のロック要求より後に到着した他トランザクションのロック要求が t より前に受けつけられるという「追い越し事象」は発生しない。一方，EXP, FICOFO ではこれが発生しうる。ただし FICOFO では，追い越していくロック要求は t のロック要求と両立するものに限られる。したがって，t のロック要求と非両立なロック要求が後から到着しても次々と先に受けつけられ，t のロック要求が長

時間受けつけられないという「沈みこみ現象」は発生しない。EXP ではこれが発生し，性能低下を招く恐れがある。

次にデッドロック発生率については，次の命題が成立する。

〔命題 3〕

FIFO, EXP, FICOFO の各ロック割当て方式のもとでのデッドロック発生率をそれぞれ P(FIFO), P(EXP), P(FICOFO) とすると，次式が成り立つ。

$$P(\text{FIFO}) \geq P(\text{FICOFO}) \geq P(\text{EXP}) \quad \cdots\cdots(13)$$

（証　明）

2.2節の命題 1′ で述べたように，リソース r にトランクション t のロック要求が到着したとき，デッドロックが発生するならば，待ちグラフ中で t と $h(\epsilon H_R)$ を含むサイクルが形成される。h から t にもどるパスは r 以外のリソースでの待ちによるので，ここでは与件とみなされる。したがって，デッドロック発生率 P は t から $h(\epsilon H_R)$ へのパスの数すなわち H_R の元の数で評価できる。いま，FIFO, EXP, FICOFO の各方式における H_R をそれぞれ H_R(FIFO), H_R(EXP), H_R(FICOFO) とかくと，方式の定義および H_R の定義より次式がえられる。

$$\begin{cases} H_R(\text{FIFO}) = H_D \cup \{h \mid \exists w \epsilon W_R(\text{FIFO}) \ s.t. \ f(m(h), m(w)) = 0\} \\ H_R(\text{EXP}) = H_D \\ H_R(\text{FICOFO}) = H_D \cup \{h \mid \exists w \epsilon W_R(\text{FICOFO}) \ s.t. \ f(m(h), m(w)) = 0\} \end{cases} \quad \cdots\cdots(14)$$

ただし式 (14) で，W_R(FIFO), W_R(FICOFO) はそれぞれ FIFO, FICOFO のもとで形成される有向エッジをたどって t から到達可能な w の集合である。FIFO では，待行列に並ぶ全ての w に t から有向エッジが形成されるので，W_R(FIFO) は r における全待ちトランザクションの集合 W に一致する。一方，FICOFO では，トランザクションのロック要求が待ち状態になったとき，既に待ち行列に並んでいたロック要求のうちで非両立なもののみに対して有向エッジが形成される。したがって当然，

$$W = W_R(\text{FIFO}) \supseteq W_R(\text{FICOFO}) \quad \cdots\cdots(15)$$

である。ゆえに，式(14), (15)より，

$$H_R(\text{FIFO}) \supseteq H_R(\text{FICOFO}) \supseteq H_R(\text{EXP}) \quad \cdots\cdots(16)$$

が成り立つ。以上より証明された。

（証明終り）

3.2 方式実現上の考察

FIFO, EXP, FICOFO の各ロック割当て方式のもとでのプログラム処理オーバヘッドについて考察する。ロック割当てのためのプログラムが動作するのは，ロック要求発生時とロッ

ク解放時の二つの場合である。ここでロック要求発生時の処理オーバヘッドについては，「グループ・モード」という概念を用いると三方式ともほぼ等しくすることができる。

グループ・モード $M(G)$ は，トランザクション集合 $G=\{g_1, g_2, \cdots\}$ について定義され，任意のトランザクション t のモード $m(t)$ と次の関係をみたす。

$$f(M(G), m(t))=\begin{cases}1 : f(m(g), m(t))=1 \quad \text{for} \quad {}^{\forall}g\in G \\ 0 : \text{上記以外}\end{cases} \quad \cdots\cdots(17)$$

いいかえると，或るトランザクションの要求（保持）ロックのモードがグループ・モードと両立することは，それが当該グループ中の全てのトランザクションの要求（保持）ロックのモードと両立することに等しい。実システムでは，グループ・モードについての両立性関数は，当該グループのメンバーの両立性関数の AND 演算をとったものとして予め用意しておくことができる。

トランザクション t のロック要求が発生したとき，FIFO では，待ちトランザクションが無い場合（$W=\phi$）にはロック保持トランザクションのグループ・モード $M(H)$ と $m(t)$ の両立性が判定される。$W\neq\phi$ なら t はただちに待ち状態になる。EXP では，待ちトランザクションの有無によらず，$M(H)$ と $m(t)$ の両立性が判定される。FICOFO では，グループ・モード $M(H\cup W)$ すなわちロック保持トランザクションと待ちトランザクションの和集合のグループ・モードと $m(t)$ の両立性が判定される。

上記のごとく，ロック要求時の判定処理オーバヘッドは三方式とも大差は無い。一方，ロック解放時のプログラム処理オーバヘッドは，三方式でかなり異なる。3.1 節で述べたように，FIFO では待行列の先頭から順にロック要求を受けつけていくが，受けつけられない要求があらわれるとそこで処理は中止される。一方，EXP と FICOFO では，受けつけられない要求があらわれても原則として待行列の最後尾まで処理が継続される。したがって処理オーバヘッドは，EXP や FICOFO が FIFO より大きい。

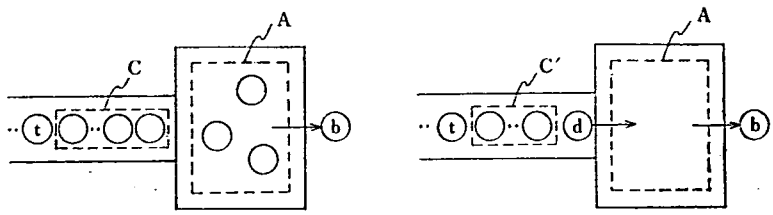
さらに実は，EXP と FICOFO とを比較すると，FICOFO では必ずしも待行列の最後尾まで処理を継続しないですむ可能性が高く，したがって多くの場合，三方式の処理オーバヘッドを Q(FIFO), Q(EXP), Q(FICOFO) とすれば，

$$Q(\text{EXP})\geq Q(\text{FICOFO})\geq Q(\text{FIFO}) \quad \cdots\cdots(18)$$

となる。以下，式 (18) の根拠について述べる。

EXP と FICOFO について，まず，途中で受けつけ不能な要求が現れても必ず待行列の最後尾まで処理の継続が必要となる条件を示す。続いて，実用上多くのシステムで用いられている 5 モード体系[1] (X, SIX, IX, S, IS) のもとで，得られた条件をみたすモードが存在するか否かについて考察を加える。

図 2 において，A はロックを保持しているトランザクションの集合，b はロックを解放し



(a) $A \neq \phi$　　(b) $A = \phi$

図 2 プログラム処理オーバヘッド算定モデル

たトランザクション，C は待行列中でトランザクション t より前に位置するトランザクションの集合とする。受けつけ不能なロック要求が現れても処理を継続する必要があるのは，C の中に 1 個以上受けつけ不能なロックを要求するトランザクションが存在し，かつ t のロック要求を受けつけることが可能な場合である。このための条件を明示するにあたり，特に A が空集合の場合は別に論ずる必要がある。すなわちこのとき，待行列の先頭のトランザクション d のロック要求は，EXP でも FICOFO でも必ず受けつけられる。いま，C から d を除いたトランザクション集合を C' とする。

(1) $A=\phi$ のとき

まず，EXP の場合について述べる。いま，b は A とともにロックを保持していたわけだから，

$$f(M(A),\ m(b))=1 \quad \cdots\cdots(19)$$

である。次に C 内のトランザクションの要求の中には少なくとも 1 個受けつけ不能なものが含まれるから，

$$f(M(A),\ M(C))=0 \quad \cdots\cdots(20)$$

である。t の要求が受けつけられるので，

$$f(M(A),\ m(t))=1 \quad \cdots\cdots(21)$$

が成り立たねばならない。また，b がロックを解放してはじめて t のロック要求が受けつけられることから，

$$f(m(b),\ m(t))=0 \quad \cdots\cdots(22)$$

が成り立つ。

一方，FICOFO では，式(19)～(22)に加え，非両立なモードのロック要求による追い越しが許されないことから，しらに次式が成り立つ。

$$f(M(C),\ m(t))=1 \quad \cdots\cdots(23)$$

(2) $A=\phi$ のとき

まず，EXP の場合について述べる。b がロックを解放するまで d のロック要求は受けつけられなかったことから，

$$f(m(b),\ m(d))=0 \quad \cdots\cdots(24)$$

である。次に C' 内のトランザクションのロック要求の中には少なくとも 1 個受けつけ不能なものが含まれるから，

$$f(M(C'),\ m(d))=0 \quad \cdots\cdots(25)$$

である。t の要求が受けつけられるので，

$$f(m(d),\ m(t))=1 \quad \cdots\cdots(26)$$

である。また，EXP のもとで，C' 内のトランザクションの要求の中には b のロック解放まで受けつけられなかったものが存在することから，

$$f(m(b),\ M(C'))=0 \quad \cdots\cdots(27)$$

が成り立つ。さらに，b がロックを解放してはじめて t のロック要求が受けつけられることから，

$$f(m(b),\ m(t))=0 \quad \cdots\cdots(28)$$

が成り立つ。

一方，FICOFO では，式(24)(25)(26)(28)に加え，非両立なモードのロック要求による追い越しが許されないことから，式(27)のかわりに式(29)が成り立つ。

$$f(M(C'),\ m(t))=1 \quad \cdots\cdots(29)$$

上記(1)，(2)の結果を表 1 にまとめる。

システムで採用されているモード体系によっては，表 1 の条件が実際には成立せず，したがっ

表 1　待行列全件処理のための一般的条件

	$A\neq\phi$	$A=\phi$
EXP	$f(M(A),\ m(b))=1$ $f(M(A),\ M(C))=0$ $f(M(A),\ m(t))=1$ $f(m(b),\ m(t))=0$	$f(m(b),\ m(d))=0$ $f(M(C'),\ m(d))=0$ $f(m(d),\ m(t))=1$ $f(m(b),\ M(C'))=0$ $f(m(b),\ m(t))=0$
FICOFO	$f(M(A),\ m(b))=1$ $f(M(A),\ M(C))=0$ $f(M(A),\ m(t))=1$ $f(m(b),\ m(t))=0$ $f(M(C),\ m(t))=1$	$f(m(b),\ m(d))=0$ $f(M(C'),\ m(d))=0$ $f(m(d),\ m(t))=1$ $f(m(b),\ m(t))=0$ $f(M(C'),\ m(t))=1$

て待行列の途中で受けつけられないロック要求が現れればその時点で処理を終了してもよいことがある。以下，実システムで多用されている5モード体系について表1の条件が成立するか否かを検討する。

本論文で扱う5モード体系は，X (Exclusive)，S (Share)，IX (Intention Exclusive)，IS (Intention Share)，SIX (Share and Intention Exclusive) からなる。X，S はそれぞれリソースに対する書きこみ，読みこみ時に用いられる。また IX，IS，SIX は階層ロック[1]のためのモードである。ここで階層ロックとは，並行処理多重度の向上とロック管理オーバヘッドの削減を目的とする方式上の特徴である。階層ロックを用いると，多量なリソースをアクセスするトランザクションにはデータベース・エリアなどの大きな単位（上位階層リソース），少量なリソースをアクセスするトランザクションにはデータベース・ページなどの小さな単位（下位階層リソース）でロックをかけることが可能である。上位階層リソースは一般に複数の下位階層リソースから構成されるので，下位階層リソースに X や S モードのロックをかけるためには，予めその上位階層リソースに，それぞれ IX や IS モードの意図 (Intention) ロックをかけなければならない。また，SIX は S と IX をあわせたモードであり，少量リソースに書きこみ，多量リソースを読みこむトランザクションに用いられる。

5モードの両立性関数をマトリックスの形で表2に示す。表2において，マトリックスの第 (i, j) 成分は $f(m_i, m_j)$ を表わす $(m_i, m_j = X, SIX, IX, S, IS)$。なお，この5モード体系についての詳細は，文献1)を参照されたい。

表2の5モード体系のもとで，表1の条件をみたすモードのくみあわせについて検討すると，それは表3に示す場合しかありえないことがわかる。証明は付録に示す。

したがって実際には，FICOFO においては，待行列の途中で受けつけられないロック要求が現れればその時点で処理を終了してもよい場合が多い。したがって，不等式(18)が成立する。

表 2　5モード体系の両立性マトリックス

	X	SIX	IX	S	IS
X	0	0	0	0	0
SIX	0	0	0	0	1
IX	0	0	1	0	1
S	0	0	0	1	1
IS	0	1	1	1	1

(0：非両立，1：両立)

表 3　待行列全件処理のための5モード体系における条件

	$A \neq \phi$	$A = \phi$
EXP	$\begin{cases} M(A) = IS \\ M(C) = X \end{cases}$	many cases (refer to Appendix)
FICOFO	no cases	$\begin{cases} m(b) = X \\ m(t) = IS \end{cases}$

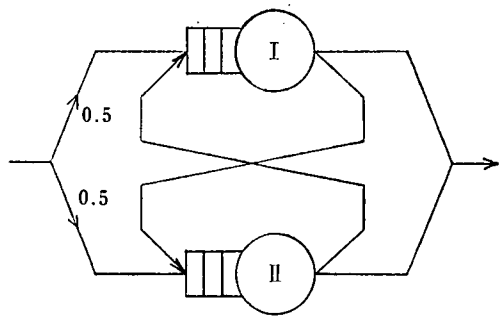


図 3　シミュレーション・モデル

4.　シミュレーション評価

シミュレーションを実行し，FIFO，EXP，FICOFO の各ロック割当て方式のもとでのロック待ち時間とデッドロック発生率を比較評価した。ロック割当て方式による差異を強調するため，図3に示すようにロック割当て機構以外の部分はできるだけ簡略化したシミュレーション・モデルを用いた。

図3において，システム内にはリソースⅠとリソースⅡの2個のリソースが存在する。発生したトランザクションは，これらリソースのロックを要求，獲得し，リソースを使用した後，消滅する。トランザクションのタイプとして，リソースⅠを使用した後リソースⅡを使用するタイプと，逆にリソースⅡを使用した後リソースⅠを使用するタイプの二つがある。前者，後者いずれのタイプになるかは，トランザクション発生時に等確率 0.5 でランダムに決定される。トランザクションの発生時間間隔は平均 30～80ms の指数分布にしたがう。またリソースのサービス時間は，リソースⅠ，Ⅱいずれも平均 50ms の一点分布にしたがう。ただし，トランザクションは実行完了するまでいったん獲得したロックを解放することは無いので，例えばリソースⅠ，リソースⅡの順序でリソースを使用するタイプのトランザクションは，リソースⅡのサービス完了までリソースⅠのロックを保有し続ける。逆に，いま一方のタイプのトランザクションは，リソースⅡのサービスが完了するとそのロックを保有したままリソースⅠのロックを要求し，リソースⅡのサービス完了後二つのロックを同時に解放する。

なお，Ⅰ，Ⅱいずれのリソースについても，トランザクションがリソースを使用するにあたって要求するロックのモードは，*X*，*SIX*，*IX*，*S*，*IS* の5モードの中から等確率 0.2 でランダムに選ばれる。トランザクションは，一方のリソースのロックを保有したまま他方のリソースのロックを要求するので，デッドロックが発生する可能性がある。或るトランザクションのロック要求によってデッドロックが発生すると，当該トランザクションは棄却される。

リソースⅠ，Ⅱにおいて，どのトランザクションのロック要求を受けつけるかは，表2の両立性マトリックスの条件をみたした上で，FIFO，EXP，FICOFO のうちいずれかのロック割当て方式により定められる。シミュレーションの測定項目としては，各ロック割当て方式のもとで，ロック要求を発行してから受けつけられるまでのモード別のロック待ち時間，および

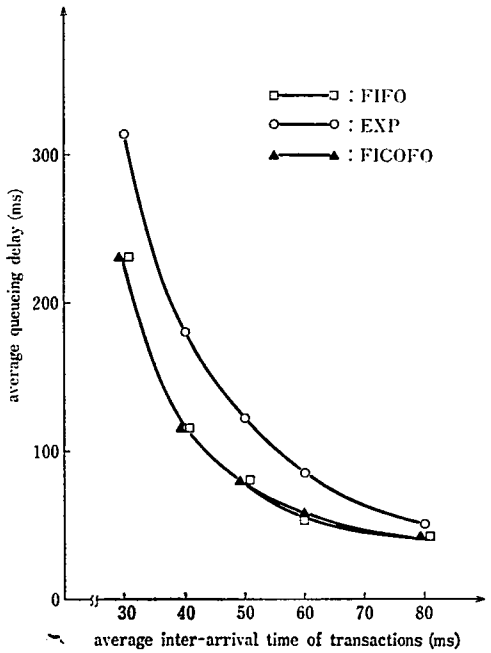


図 4 *X* モード・ロックの平均待ち時間

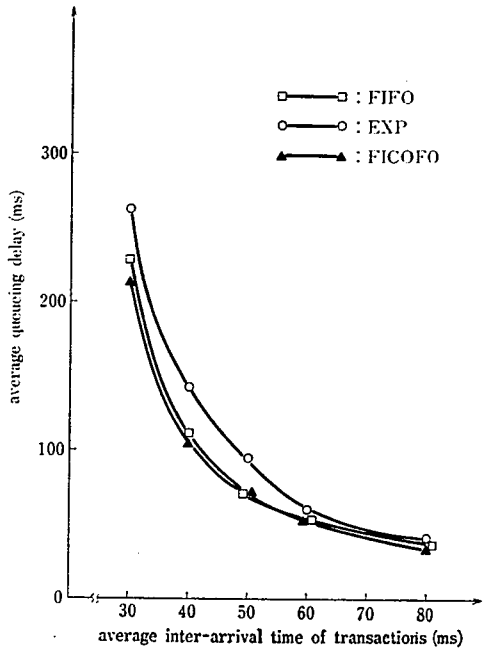


図 5 *SIX* モード・ロックの平均待ち時間

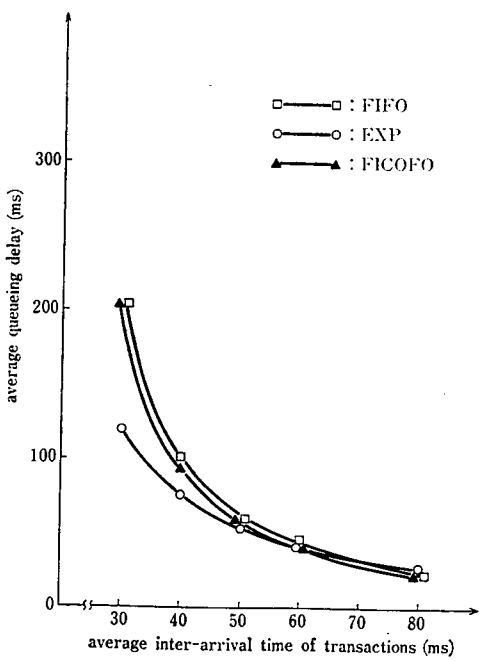


図 6 *IX* モード・ロックの平均待ち時間

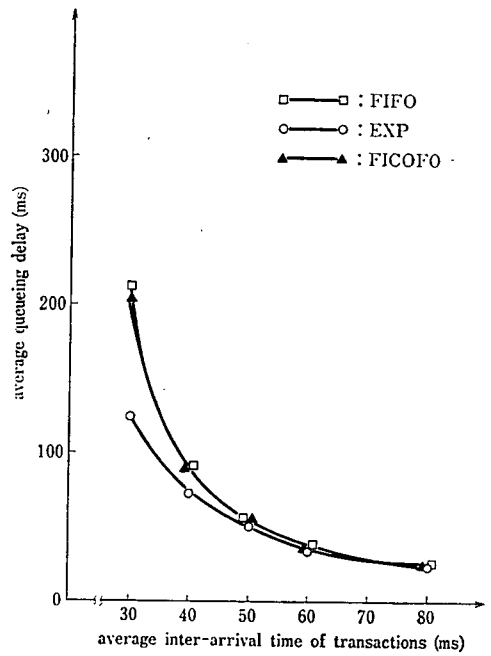


図 7 *S* モード・ロックの平均待ち時間

デッドロック発生率を採択した。

図4～図8は，それぞれ *X*, *SIX*, *IX*, *S*, *IS* の各モードでロックを要求したときの平均待ち時間である。また図9は，デッドロック発生率（＝100＊デッドロック発生回数/ロック要

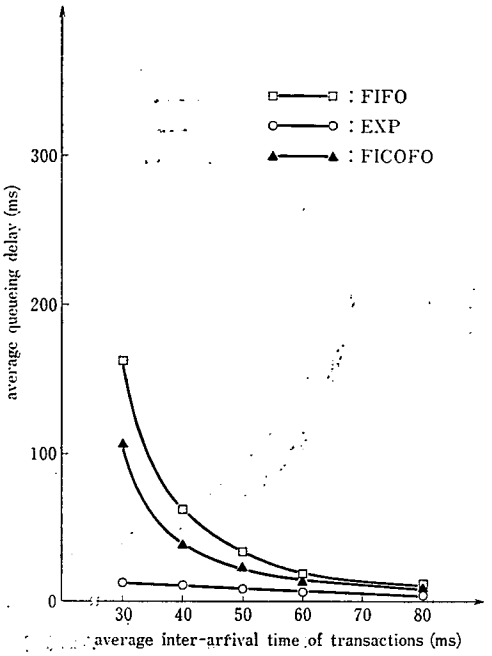


図 8　*IS* モード・ロックの平均待ち時間

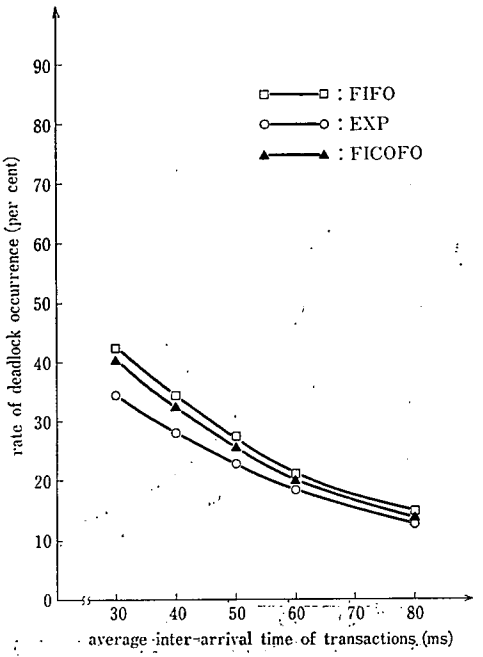


図 9　デッドロック発生率

求回数）である。当然のことながら，方式による差異はトランザクションの発生頻度とともに増大する。まずロック待ち時間に着目すると，FIFO と FICOFO は *IS* 以外の各モードについてはほぼ同程度である。他モードとの両立性の高い *IS* モードについては，FICOFO の方が FIFO より受けつけられる確率が高いのでロック待ちが減少している。EXP におけるロック待ち時間を FIFO や FICOFO におけるそれと比較すると，他モードとの両立性の高い *IS*, *S*, *IX* モードにおいて，受けつけられる確率が高いのでロック待ちが減少している。この傾向は *IS* モードにおいて最も顕著である。一方，他モードとの両立性の低い *X* や *SIX* モードでは，逆に FIFO や FICOFO よりロック待ちが増大している。

デッドロック発生率については，式 (13) に示したように，EXP が最も小さく，ついで FICOFO であり，FIFO が最も大きい。この差異は人為デッドロックによるものである。ただし，図 4 によっても示唆されるように，EXP においては追い越しが発生するので，両立性の低いモードのロック要求がいつまでも受けつけられないという沈みこみ現象が発生する可能性がある。

5. むすび

データベースの論理的一貫性を保つために用いられるロックの割当て方式として，従来より用いられる FIFO, EXP に加え，新たに FICOFO をも導入し，三方式のもとでのロック待ち時間，デッドロック発生率，プログラム処理オーバヘッドを比較評価した。

このためまず，デッドロックを検知するための待ちグラフ (wait-for graph) の一般的モデルを与えた。この待ちグラフ・モデルを用いると，通常のデッドロックに加え，ロック割当て

の方式に依存して発生する人為デッドロックをも検知することができる。この人為デッドロックは，従来の待ちグラフによっては検知できなかったものである。次に，三方式のデッドロック発生率，プログラム処理オーバヘッドについて検討した。さらにシミュレーションを実行し，三方式のもとでのロック待ち時間を比較評価した。比較にあたっては，実システムで多用される X, SIX, IX, S, IS の5モード・ロック系を仮定した。

この結果，デッドロック発生率は EXP が最も小さく，ついで FICOFO，最大が FIFO であった。一方，方式実行にともなうプログラム処理オーバヘッドは，逆に FIFO が最も小さく，ついで FICOFO, EXP の順に増大することが判明した。また，ロック待ち時間については，両立性の低いモードのロックについては FICOFO や FIFO の方が EXP より待ち時間が減少している。一方，両立性の高いモードのロックについては，EXP が最も待ち時間が小さく，ついで FICOFO であり，FIFO が最大であることが示された。

付　録

(1) EXP, $A \neq \phi$ のとき

まず，式(19)，(21)より，X は全てのモードと非両立なことから，次式が成り立つ。

$$M(A) \neq X \quad \cdots\cdots(30)$$

$$m(b) \neq X \quad \cdots\cdots(31)$$

$$m(t) \neq X \quad \cdots\cdots(32)$$

ここで

$$M(A) \neq IS$$

と仮定すると

$$M(A) = S \text{ または } IX \text{ または } SIX \quad \cdots\cdots(33)$$

となる。式(22)，(31)，(32)より，

$$m(b) \neq IS \quad \cdots\cdots(34)$$

$$m(t) \neq IS \quad \cdots\cdots(35)$$

が成り立つ。いま，式(33)で $M(A)=S$ と仮定すると，式(19)，(21)，(34)，(35)より，

$$m(b)=m(t)=S \quad \cdots\cdots(36)$$

となるが，式(36)は式(22) と矛盾する。次に，$M(A)=IX$ と仮定すると，式(19)，(21)，(34)，(35)より，

$$m(b)=m(t)=IX \quad \cdots\cdots(37)$$

となり，式(37)は式(22)と矛盾する。最後に $M(A)=SIX$ と仮定すると，式(19)，(21)をみたすためには，

$$m(b)=m(t)=IS \quad \cdots\cdots(38)$$

でなければならないが，これは式(34)，(35)に矛盾する。以上より，結局式(33)は成立せず，

次式が成り立たねばならないことがわかる。

$$M(A)=IS \cdots\cdots(39)$$

式(20)と式(39)より，

$$M(C)=X \cdots\cdots(40)$$

が導かれる。ここで式(31)，(32)より，式(22)をみたす $m(b)$，$m(t)$ の組合せは下記6通りとなる。

$$(m(b), m(t))=(IX, S), (S, IX), (SIX, IX), (IX, SIX), (SIX, S), (S, SIX) \cdots(41)$$

(2) EXP，$A=\phi$ のとき

まず，式(26)より X は全てのモードと非両立なことから次式が成り立つ。

$$m(d) \fallingdotseq X \cdots\cdots(42)$$

$$m(t) \fallingdotseq X \cdots\cdots(43)$$

式(24)，(42)より，

$$m(b) \fallingdotseq IS \cdots\cdots(44)$$

である。また式(25)，(42)より，

$$M(C') \fallingdotseq IS \cdots\cdots(45)$$

である。以下，(a)～(d)の四つの場合に分けて，式(24)～(28)をみたす組合せを求める。

(a) $m(b)=M(C')=X$ のとき

式(26)，(42)，(43)をみたす組合せは下記の9通りである。

$$(m(d)=m(t))=\begin{cases}(IS, SIX), (SIX, IS), (IS, IX), (IX, IS), (IS, S), \\ (S, IS), (IS, IS), (IX, IX), (S, S)\end{cases} \cdots\cdots(46)$$

(b) $m(b)=X$，$M(C') \fallingdotseq X$ のとき

式(25)より，

$$m(d) \fallingdotseq IS \cdots\cdots(47)$$

である。式(47)より，式(26)をみたす組合せは，

$$(m(d),\ m(t))=\begin{cases}(S, S), (IX, IX), (SIX, IS), \\ (IX, IS), (S, IS)\end{cases} \cdots\cdots(48)$$

の5通りである。式(25)，(48)より，結局求める組合せは下記の11通りとなる。

$$(m(d),\ M(C'),\ m(t)) = \begin{cases}(S, SIX, S), (S, IX, S), (IX, SIX, IX), (IX, S, IX), \\ (SIX, SIX, IS), (SIX, IX, IS), (SIX, S, IS), (IX, SIX, IS), \\ (IX, S, IS), (S, SIX, IS), (S, IX, IS)\end{cases} \cdots(49)$$

(c) $m(b) \fallingdotseq X$，$M(C')=X$ のとき

式(24)より，

$$m(d) \fallingdotseq IS \quad \cdots\cdots(50)$$

であり，また式(28)より

$$m(t) \fallingdotseq IS \quad \cdots\cdots(51)$$

である。式(50)，(51)より，式(26)をみたす組合せは，

$$(m(d),\ m(t)) = (S,\ S),\ (IX,\ IX) \quad \cdots\cdots(52)$$

の2通りとなる。式(24)，(28)，(52)より，結局求める組合せは下記4通りとなる。

$$(m(b),\ m(d),\ m(t)) = \begin{cases} (SIX,\ S,\ S),\ (IX,\ S,\ S), \\ (SIX,\ IX,\ IX),\ (S,\ IX,\ IX) \end{cases} \quad \cdots\cdots(53)$$

(d) $m(b) \fallingdotseq X$, $M(C') \fallingdotseq X$ のとき

前項(c)と同様にして，このときも式(50)，(51)，(52)が成り立ち，式(53)が得られる。式(25)，(27)，(53)より，求める組合せは下記6通りとなる。

$$\begin{aligned} &((m(b),\ m(d),\ M(C'),\ m(t)) \\ &= \begin{cases} (SIX,\ S,\ SIX,\ S),\ (SIX,\ S,\ IX,\ S),\ (IX,\ S,\ SIX,\ S), \\ (SIX,\ IX,\ SIX,\ IX),\ (SIX,\ IX,\ S,\ IX),\ (S,\ IX,\ SIX,\ IX) \end{cases} \end{aligned} \quad \cdots\cdots(54)$$

(3) FICOFO, $A \fallingdotseq \phi$ のとき

まず，式(19)，(21)，(23)より，X は全てのモードと非両立なことから次式が成り立つ。

$$M(A) \fallingdotseq X \quad \cdots\cdots(55)$$

$$m(b) \fallingdotseq X \quad \cdots\cdots(56)$$

$$M(C) \fallingdotseq X \quad \cdots\cdots(57)$$

$$m(t) \fallingdotseq X \quad \cdots\cdots(58)$$

以下，(a)～(d)の4つの場合に分けて，式(19)～(23)をみたす組合せが存在しないことを示す。

(a) $M(A) \fallingdotseq M(C)$ かつ $m(b) \fallingdotseq m(t)$ のとき

式(19)，(21)，(56)，(58)より，$M(A)$ は X 以外の相異なる2つのモードと両立するから，

$$M(A) \fallingdotseq SIX \quad \cdots\cdots(59)$$

である。ここで，式(20)，(57)より，

$$M(A) \fallingdotseq IS \quad \cdots\cdots(60)$$

であり，式(55)，(59)，(60)より次式が得られる。

$$M(A) = S \text{ または } IX \quad \cdots\cdots(61)$$

いま，(a)の前提より $m(b) \fallingdotseq m(t)$ である。したがって式(19)，(21)より，$M(A)$ が S, IX, いずれの場合でも，$m(b)$ が $m(t)$ のいずれか一方は IS でなくてはならない。しかし，式(56)，(58)を考えると，これは式(22)に矛盾する。

(b)　$M(A)=M(C)$ かつ $m(b)\fallingdotseq m(t)$ のとき，

このとき，式(19)，(21)，(56)，(58)より，$M(A)(=M(C))$ は X 以外の相異なる二つのモードと両立するから，

$$M(A)(=M(C))\fallingdotseq SIX \quad \cdots\cdots(62)$$

である。また，式(20)より，

$$M(A)(=M(C))\fallingdotseq IS \quad \cdots\cdots(63)$$

であり，式(55)を考えると次式が得られる。

$$M(A)(=M(C))=S \text{ または } IX \quad \cdots\cdots(64)$$

式(64)は，式(20)に矛盾する。

(c)　$M(A)\fallingdotseq M(C)$ かつ $m(b)=m(t)$ のとき

このとき，式(19)，(23)，(55)，(57)より，$m(b)(=m(t))$ は X 以外の相異なる二つのモードと両立するので，

$$m(b)(=m(t))\fallingdotseq SIX \quad \cdots\cdots(65)$$

である。また，式(22)より，

$$m(b)(=m(t))\fallingdotseq IS \quad \cdots\cdots(66)$$

であり，式(56)を考えると次式が得られる。

$$m(b)(=m(t))=S \text{ または } IX \quad \cdots\cdots(67)$$

式(67)は式(22)に矛盾する。

(d)　$M(A)=M(C)$ かつ $m(b)=m(t)$ のとき

このとき，式(20)，(22)より，

$$f(M(A),\ M(A))=f(m(b),\ m(b))=0 \quad \cdots\cdots(68)$$

が成り立つ。したがって，式(55)，(56)より，

$$M(A)=m(b)=SIX \quad \cdots\cdots(69)$$

が必要。しかし式(69)は式(19)に矛盾する。

(4)　FICOFO，$A=\phi$ のとき

まず式(26)，(29)より，X は全てのモードと非両立なことから次式が成り立つ。

$$m(d)\fallingdotseq X \quad \cdots\cdots(70)$$

$$M(C')\fallingdotseq X \quad \cdots\cdots(71)$$

$$m(t)\fallingdotseq X \quad \cdots\cdots(72)$$

また式(24)より，X 以外のモードである $m(d)$ と非両立なことから，

$$m(b)\fallingdotseq IS \quad \cdots\cdots(73)$$

である。同様に式(25)，(70)，(71)より次式が成り立つ。

$$m(d)\fallingdotseq IS \quad \cdots\cdots(74)$$

$$M(C') \fallingdotseq IS \quad \cdots\cdots(75)$$

以上の準備のもとに，(a)，(b) 二つの場合に分けて論ずる。

(a) $m(b) \fallingdotseq X$ のとき

このとき，条件をみたす組合せが存在しないことを以下に示す。まず式(28)より，

$$m(t) \fallingdotseq IS \quad \cdots\cdots(76)$$

である。ここで，$m(d)$ と $M(C')$ の関係に着目する。

(i) $m(d) \fallingdotseq M(C')$ のとき

式(26)，(29)より，$m(t)$ は相異なる二つのモードと両立するので，式(76)を考えると次式が成り立つ。

$$m(t) = S \text{ または } IX \cdots\cdots(77)$$

しかし，$m(t)$ が S，IX，いずれの場合も，式(26)，(29)が成り立つためには $m(d)$，$M(C')$ のいずれか一方は IS でなければならない。これは式(74)，(75)に矛盾する。

(ii) $m(d) = M(C')$ のとき

このとき式(25)より，式(70)，(71)を考えると，

$$m(d)(=M(C')) = SIX \cdots\cdots(78)$$

である。ここで式(26)より，

$$m(t) = IS$$

が導かれるが，これは式(76)に矛盾する。

(b) $m(b) = X$ のとき

このとき，条件をみたす組合せは以下のように7通り存在する。

(i) $m(d) \fallingdotseq M(C')$ のとき

式(26)，(29)より，$m(t)$ は相異なる二つのモードと両立するので，次式が成り立つ。

$$m(t) = S \text{ または } IX \text{ または } IS \cdots\cdots(79)$$

しかし，(a)(i)の場合と同様にして，$m(t)$ が S や IX の場合は存在しない。したがって，

$$m(t) = IS \quad \cdots\cdots(80)$$

が導かれる。このとき，式(25)，(70)，(71)より，$m(d)$，$M(C')$ の組合せは下記の6通りである。

$$(m(d),\ M(C')) = \begin{cases} (SIX,\ IX),\ (SIX,\ S),\ (IX,\ SIX), \\ (IX,\ S),\ (S,\ SIX),\ (S,\ IX) \end{cases} \quad \cdots\cdots(81)$$

(ii) $m(d) = M(C')$ のとき

このとき式(25)より，式(70)，(71)を考えると，

$$m(d)(=M(C')) = SIX \quad \cdots\cdots(82)$$

である。ここで式(26)より，

$$m(t)=IS \quad \cdots\cdots(83)$$

となる。

参 考 文 献

1) J. N. Gray: Notes on Database Operating Systems, in "Operating Systems—An Advanced Course (Eds: R. Bayer et al.)", Springer-Verlag (Berlin), pp. 393–481, 1978.

2) D. J. Rypka, and A. P. Lucido: Deadlock Detection and Avoidance for Shared Logical Resources, IEEE Trans. Softw. Eng., Vol. SE-5, No. 5, pp. 465–471, 1979.

3) R. C. Holt: Comments on Prevention of System Deadlocks, Comm. ACM, Vol. 14, No. 1, pp. 36–38, 1971.

4) C. J. Date: An Introduction to Database Systems, Vol. II, Addison-Wesley (Massachusetts), 1983.

5) P. F. King, and A. J. Collmeyer: Database sharing—An efficient mechanism for supporting concurrent processes, Proc. NCC 73, pp. 271–275, 1973.

6) D. A. Menasce, and R. R. Muntz: Locking and Deadlock Detection in Distributed Data Bases, IEEE Trans. Softw. Eng., Vol. SE-5, No. 3, pp. 195–202, 1979.

7) P. A. Bernstein, and N. Goodman: Concurrency Control in Distributed Database Systems, Comput. Surv., Vol. 13, No. 2, pp. 185–221, 1981.